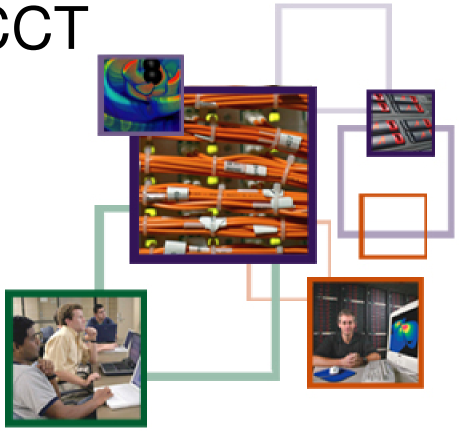
Cactus Tutorial

Introduction to Cactus

Yaakoub El Khamra

Cactus Developer, Frameworks Group CCT

27 March, 2007



Center for Computation & Technology

AT LOUISIANA STATE UNIVERSITY

Agenda

- Introduction to Cactus
  - What is Cactus
  - Flesh and thorns
  - Cactus Computational Toolkit
- Cactus Demos
  - WaveToy Demo

What is Cactus

- Cactus is a framework for developing portable, modular applications, in particular, although not exclusively, high-performance simulation codes.
- Cactus is designed to allow experts in different fields to develop modules based upon their expertise and to leverage off modules developed by experts in other fields to perform their work, with minimal knowledge of the internals or operation of the other modules.

What is a Framework

- “ a framework is a re-usable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact”
- “a framework is the skeleton of an application that can be customized by an application developer”
- “a framework is an architecture, plus an implementation, plus documentation that captures the intended use of the framework for building applications”

Other Frameworks

- Many framework-like tools developed over the last few years
- POOMA
- Overture
- SAMRAI
- KeLP
- PETSc
- HYPRE
- Trilinos
- Common Component Architecture
- ...

In a Nutshell...

- Cactus acts a the “main” routine of your code, it takes care of e.g. parallelism, IO, checkpointing, parameter file parsing for you (if you want), and provides different computational infrastructure such as reduction operators, interpolators, coordinates, elliptic solvers, …
- Everything Cactus “does” is contained in thorns (modules), which you need to compile-in. If you need to use interpolation, you need to find and add a thorn which does interpolation.
- It is very extensible, you can add you own interpolators, IO methods etc.
- Not all the computational infrastructure you need is necessarily there, but hopefully all of the APIs etc are there to allow you to add anything which is missing, and the community is always adding
- We’re trying to provide a easy-to-use environment for collaborative, high-performance computing, from easy compilation on any machine, to easy visualization of your output data.

Cactus Goals

- Portable
  - Must be able to compile and run on any platform we want to run on
- Modular
  - People should be able to write modules which interact through standard interfaces with other modules without having to know the internals of the other modules
  - Modules with same functionality should be interchangeable

Cactus Goals...(cont)

- Support legacy codes
  - Old codes must be able to become modules without significant changes
  - Must support Fortran 77 !
  - Should have data structures compatible with old codes
- Make use of existing technologies and tools where appropriate

Cactus Goals....(cont)

- Future proof
  - Must not be tied to any particular paradigm
  - Parallelism should be independent but compatible with currently available systems such as MPI or PVM
  - IO system should be independent of but compatible with currently available systems such as HDF
  - Support current and future programming languages

Cactus Goals... (cont)

- Easy to use
  - If not, it won't be used
  - Must be well documented for users and developers
  - Should allow people to program as before.
- Maintainable
  - Internals should be cleanly written
  - Internals should be documented

Current Users

AEI

Southampton

Wash U

RIKEN

Chemical Engineering (U.Kansas)

LSU

TAC

Penn State

Thessaloniki

JPL

Austin

Goddard

SISSA

Portsmouth

Climate Modeling (NASA, Netherlands)

EU Astrophysics Network

NASA Neutron Star Grand Challenge

Bio-Informatics (LSU, Bio-Grid, Canada)

Tuebingen

Astrophysical Hydrodynamics (LSU)

Geophysics (Stanford)

Brownsville

Early Universe (LBL)

Quantum Gravity (Hamilton)

German SFB Consortium

CFD (LSU)

Damage Mechanics (LSU)

Crack Prop. (Cornell)

Astrophysics (Zeus)

Current Developers

LSU Cactus Team

Washington University

Cactus User Community

Clemson

TAC

University Of Kansas

Argonne National Laboratory

Lawrence Berkeley Laboratory

Albert Einstein Institute

Hamilton College

GridLab

NCSA

Konrad-Zuse Zentrum

Tuebingen

EGrid

Compaq

Sun

University Of Chicago

Microsoft

SGI

Intel

External Funding:

NSF, Max Planck Gesellschaft, Deutsches Forschungsnetz, DOE, NASA, NCSA, European Commission

Philosophy

- Open code base and community contributions crucial
- Strict quality control for base framework
- Development always driven by real users requirements
- Application driver for computer science projects
- Leverage other projects where possible
- Support and develop for a wide range of application domains
- Provide tools for a complete working environment

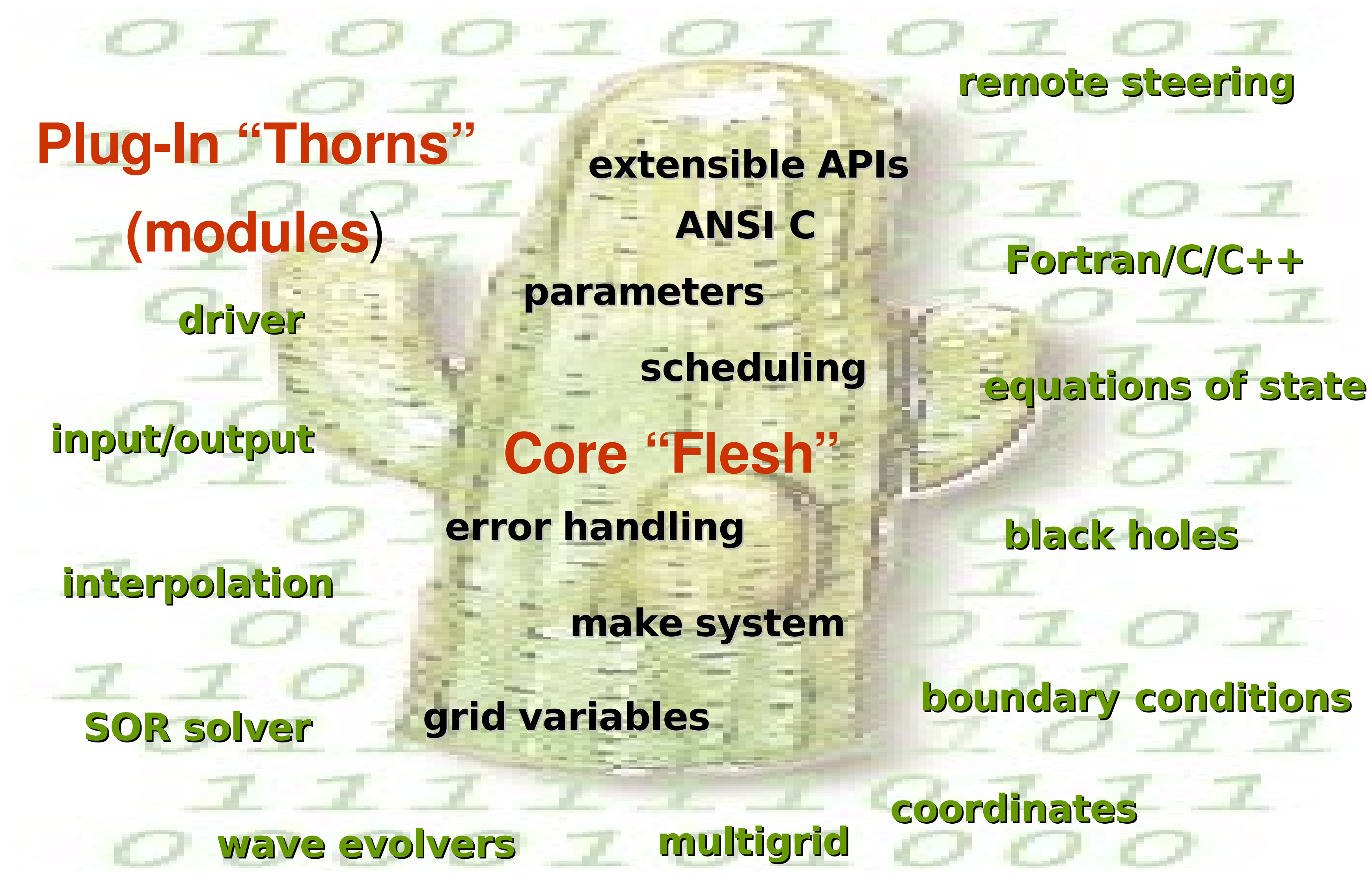
Agenda

- Introduction to Cactus
  - What is Cactus
  - Flesh and thorns
  - Cactus Computational Toolkit
- Cactus Demo
  - WaveToy Demo

Structure

- The source code of Cactus consists of a core part – the “Flesh” and a set of modules called “thorns”.
- The Flesh is independent of all thorns and after Cactus is initialized, it generally acts as a utility and service library which the thorns call to get information or ask for some action to happen.
- Thorns are separate libraries which encapsulate some functionality. In order to keep a distinction between functionality and implementation of the functionality, each thorn declares that it provides a certain “implementation”. Different thorns can provide the same “implementation”, and thorn dependencies are expressed in terms of “implementations” rather than explicit references to thorns, thus allowing the different thorns providing the same “implementation” to be interchangeable.

Structure....



Plug-In “Thorns”

(modules)

extensible APIs

ANSI C

parameters

scheduling

Core “Flesh”

error handling

make system

grid variables

remote steering

Fortran/C/C++

driver

equations of state

input/output

black holes

interpolation

boundary conditions

SOR solver

coordinates

wave evolvers

multigrid

The Flesh

- Make System
  - Organizes builds as *configurations* which hold everything needed to build with a particular set of options on a particular architecture.
- API
  - Functions which must be there for thorns to operate.
- Scheduling
  - Sophisticated scheduler which calls thorn-provided functions as and when needed.
- CCL
  - Configuration language which tells the flesh all it needs to know about the thorns.

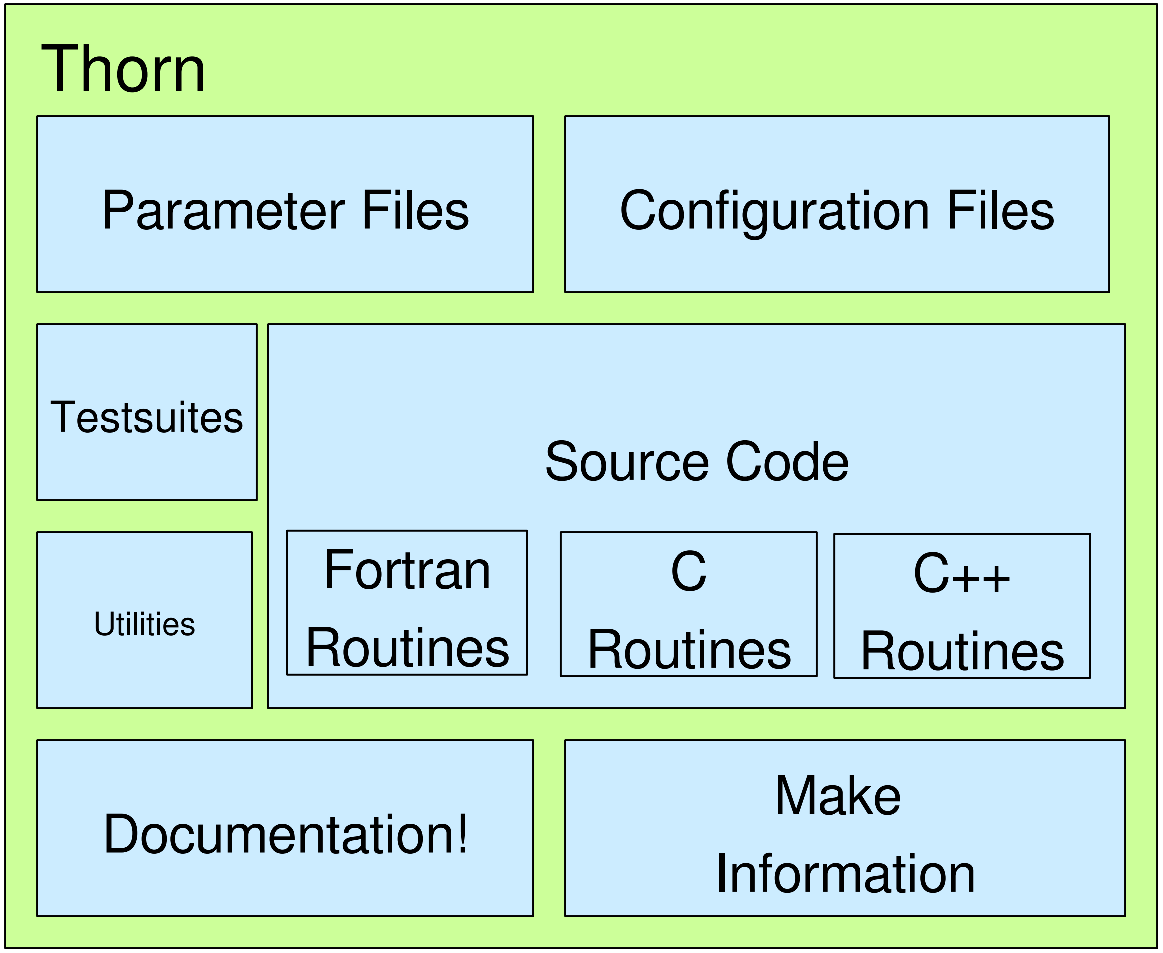
The make system

- Designed to allow same source checkout to be used to build on various architectures and for various compilation options on one machine.
- Compilation options grouped into **configuration time** options and **compile time** options.
  - Configuration time - compilers, optimization and debugging flags, etc.
  - Compilation time - warnings, verbosity of compilation, etc.
- Each configuration has a **ThornList** which lists the thorns to be compiled in. When this list changes, only those thorns directly affected by the change are recompiled.

Thorn Specification

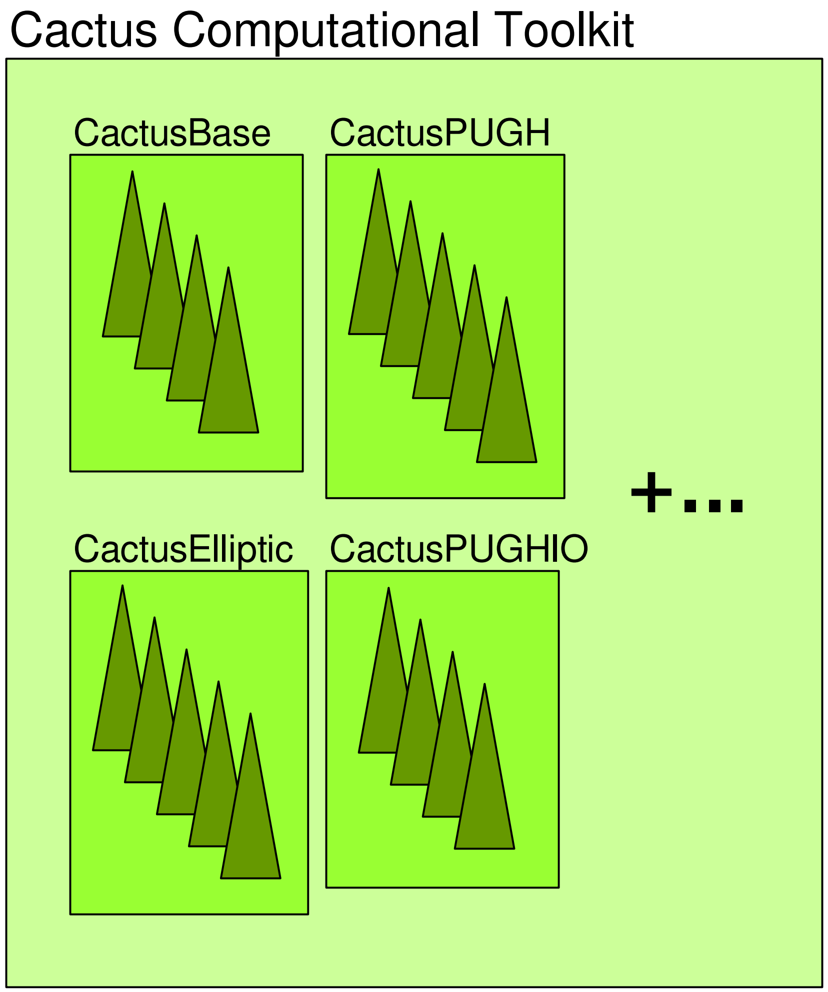
- The Flesh finds out about thorns by configuration files in each thorn. These files are converted at compile time into a set of routines the Flesh can call to find out about thorns.
- There are three such files
  - Scheduling directives
    - The flesh incorporates a scheduler which is used to call defined routines from different thorns in a particular order.
  - Interface definitions
    - All variables which are passed between scheduled routines need to be declared.
    - Any thorn-provided functions which other thorns call should be declared.
  - Parameter definitions
    - The flesh and thorns are controlled by a parameter file; parameters must be declared along with their allowed values.

More about Thorns



More about Thorns

- For organizational convenience, thorns are grouped into *arrangements*
  - may have related functionality (e.g. IO or Maxwell solvers)
  - may have the same author
  - may contain everything needed for one problem
- We call a collection of arrangements, a toolkit e.g.
  - Cactus Computational Toolkit
  - Cactus Relativity Toolkit



Scheduling

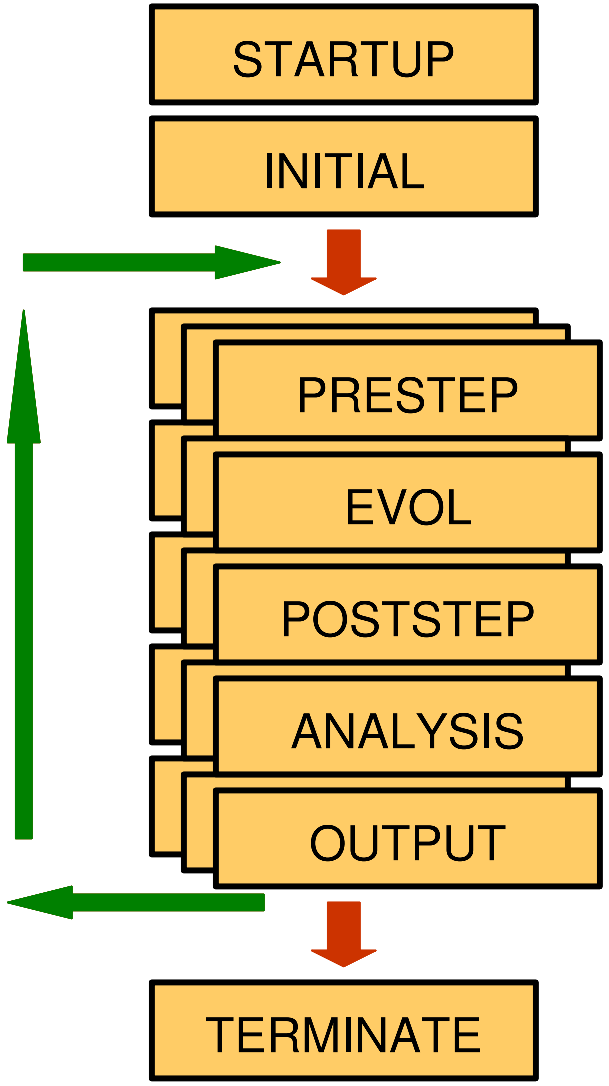
- Defined in schedule.ccl
- The Cactus Flesh contains a flexible rule based scheduler which controls the program flow.
- The scheduler calls routines from thorns, the order in which the routines are called is prescribed in a thorn configuration file.
- Scheduler also takes care of telling the Driver to assign memory for or to synchronize variables
- (The main calling routines in the Flesh are also overloadable, providing a flexible and configurable mechanism for customising program flow)

Scheduling

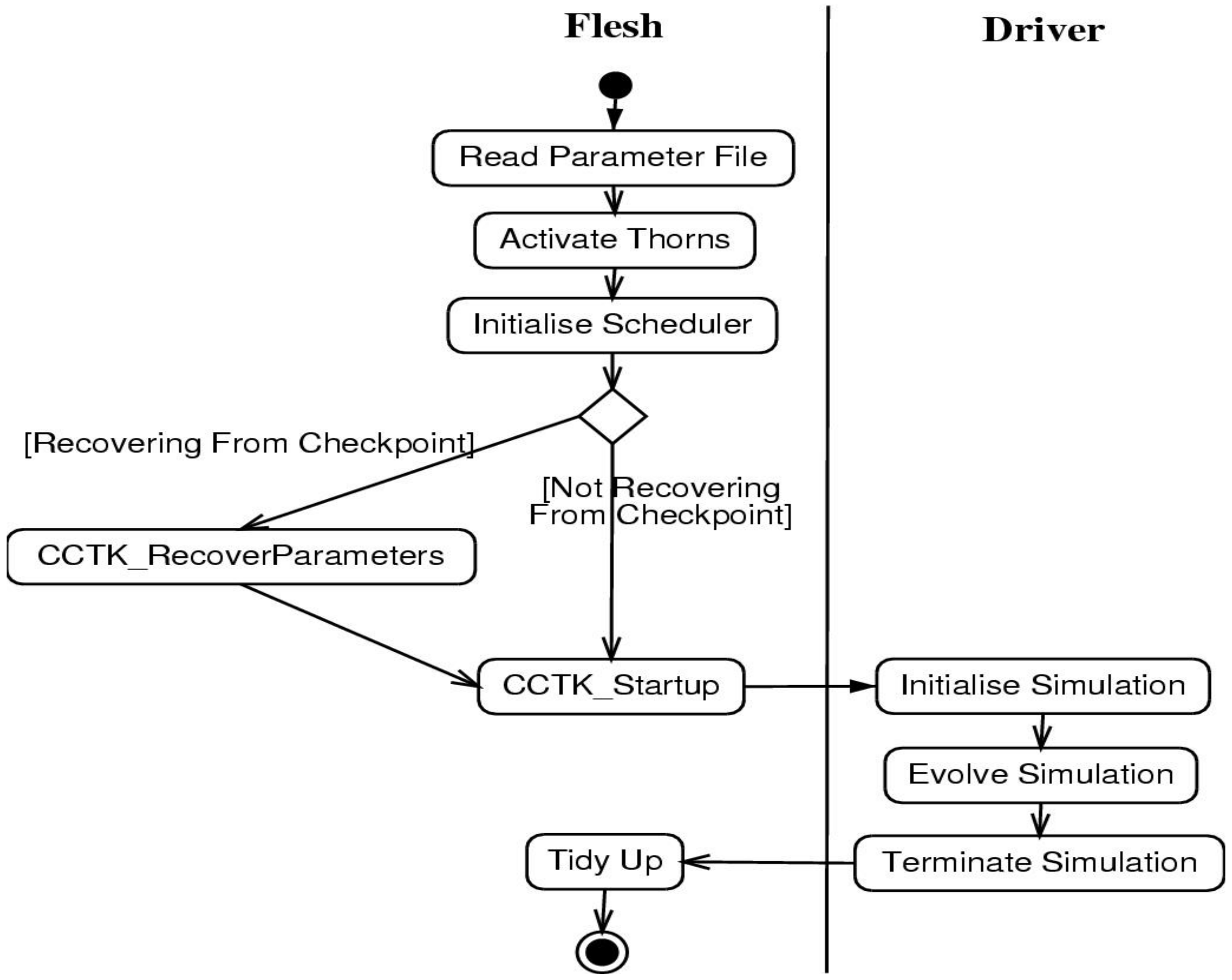
- Thorns specify which functions are to be called at which time, and in which order.
- Rule based scheduling system
- Routines are either **before** or **after** other routines (or don't care).
- Routines can be grouped, and whole group scheduled.
- Functions or groups can be scheduled **while** some condition is true.
- Flesh sorts all rules and flags an error for inconsistent schedule requests.

Standard Scheduling Tree

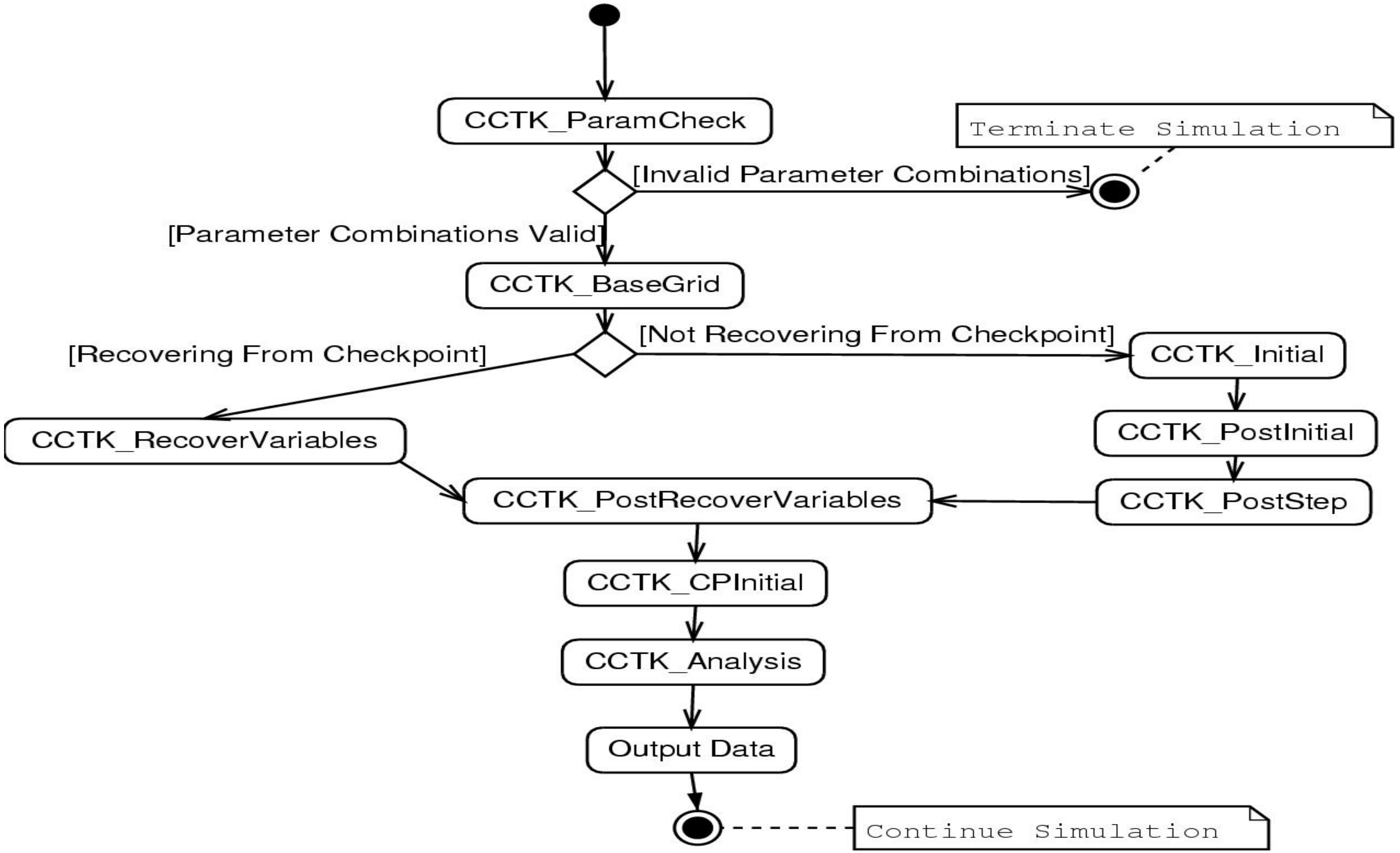
- Thorns typically register their routines to be run in one of the standard time bins
- Can define own time bins
- Many additional features: while loops, schedule as etc.
- Scheduling dependent on parameters … want to develop more flexible script based system.



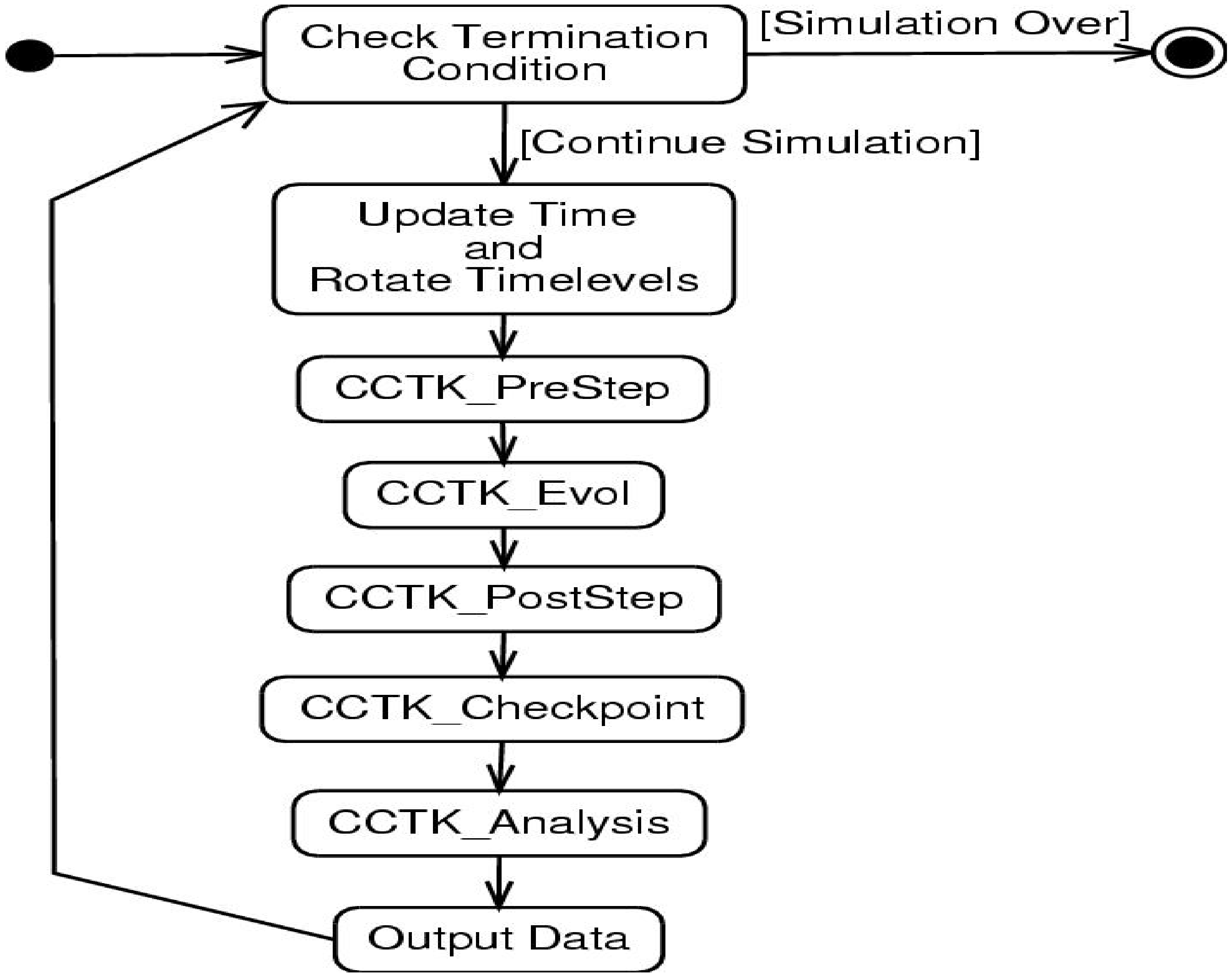
Scheduling: Program Flow



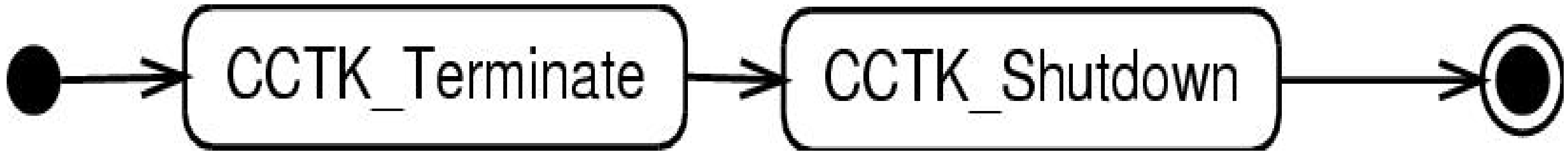
Simulation Initialization



Evolution


Check Termination Condition
[Simulation Over]
[Continue Simulation]
Update Time and Rotate Timelevels
CCTK_PreStep
CCTK_Evol
CCTK_PostStep
CCTK_Checkpoint
CCTK_Analysis
Output Data

Termination



The Driver Layer

- In principle drivers are the only thorns which know anything about parallelism
- Other thorns access parallelism via an API provided by the flesh
- Underlying parallel layer could be anything from a TCP-socket to Java RMI. It should be transparent to application thorns.
- Could even be a combination of things.
- Can even run with no parallel layer at all.
- Can pick actual driver to use at runtime - no need to recompile code to test differences between parallel layers. Can take one executable and use whatever the best layer for any particular environment happens to be.

Variables

- Defined in interface.ccl
- Essentially these are just variables which your thorn tells the Cactus infrastructure about, to get parallelisation, IO, interpolation, communication, checkpointing etc.
- Public, restricted or private, depending on who should see them.
- Can be arbitrarily sized grid arrays, or grid functions fixed to the size of the computational domain, or grid scalars.
- Many other available features: any dimension, distribution type etc.

Variables: Current Drivers

- There are several drivers available at the moment, both developed by the cactus team and by the community.
- **PUGH:** a parallel uni-grid driver, which comes as part of the the computational toolkit
- **PAGH:** a parallel AMR driver which uses the GrACE library for grid hierarchy management
- **Carpet:** a parallel fixed mesh refinement driver
- **SimpleDriver**: a simple demonstration driver which illustrates driver development
- **OAK:** a parallel tree driver (under development)
- **TAKA:** Samrai-based AMR driver
- Work has begun on other drivers: HYPRE, MOAB among others to support application development (CFD, reservoir simulations etc...)

Parameters

- Defined in param.ccl
- Range checking and validation
  - combats old problem of setting parameters to values which didn’t exist:

    evolution_method = “super stable fast 10th order shock capturing”
  - thorn writers must now specify ranges and descriptions for all parameters
  - checked at run time
- Steerable
  - new implementation of steerable/changeable parameters for remote steering
  - must define steerable … only if it makes sense to do so

Agenda

- Introduction to Cactus
  - What is Cactus
  - Flesh and thorns
  - Cactus Computational Toolkit
- Cactus Demo
  - CactusWave Demo

The Computational Toolkit

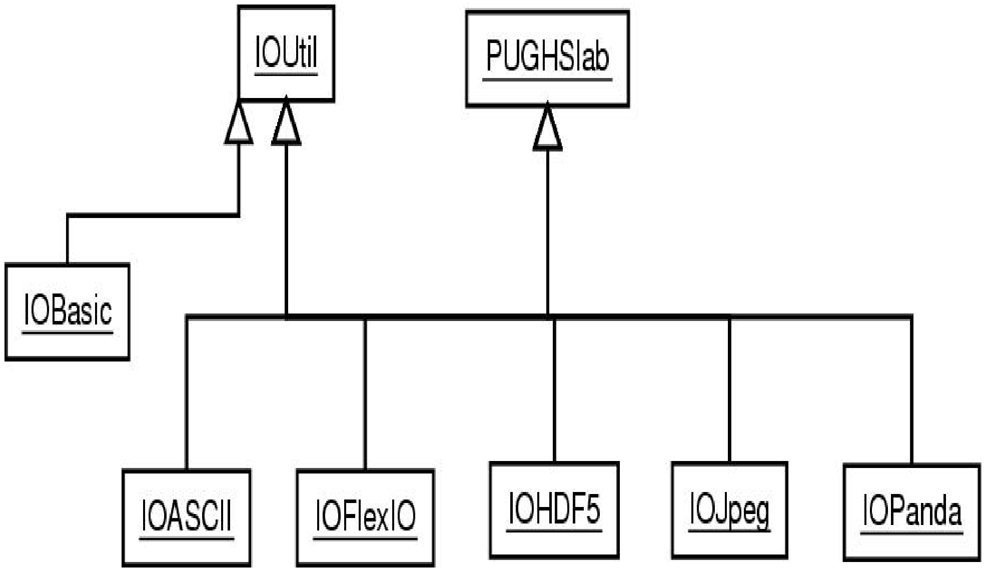
- Core thorns which provide many basic utilities, such as:
  - Boundary conditions
  - I/O methods
  - Reduction and Interpolation operations
  - Coordinate Symmetries
  - Parallel drivers
  - Elliptic solvers
  - Web-based interaction and monitoring interface
  - ...

The Computational Toolkit

- CactusBase
  - Boundary, IOUtil, IOBasic, CartGrid3D, IOASCII, Time, LocalInterp, LocalReduce, SymBase
- CactusBench
  - BenchADM, BenchIO
- CactusConnect
  - HTTPD HTTPDExtra, Socket
- CactusExamples
- CactusElliptic
  - EllBase, EllPETSc, EllSOR, EllTest
- CactusPUGH
  - PUGHInterp, PUGH, PUGHReduce, PUGHSlab
- CactusPUGHIO
  - IOFlexIO, IOHDF5, IOPanda, IOStreamedHDF5, IsoSurfacer
- CactusIO
  - IOJpeg
- CactusUtils
  - NANChecker
- CactusWave
  - IDScalarWave, IDScalarWaveC, IDScalarWaveCXX, WaveBinarySource, WaveToyC, WaveToyCXX, WaveToyF77, WaveToyF90, WaveToyFreeF90
- CactusExternal
  - FlexIO, jpeg6b

Current Capabilities: IO

- Support for IO and checkpointing in many different formats
  - Basic screen output of norms
  - 2-d slices as jpegs.
  - n-d ASCII data suitable for x/y-graph or gnuplot.
  - n-d data in John Shalf's IEEEIO format.
  - n-d data in HDF5 format, which may be written to disk or streamed to visualisation clients or other simulations.
  - Output using the Panda software from UIUC.



Current Capabilities: Grids, Boundaries, Symmetries...

- Cactus currently supports data on structured meshes. These meshes can either be unigrid, or can be adapted in either fixed mesh refinement or adaptively.
- The grid can be restricted to octants, quadrants or “bitants”.
- The thorns provided with Cactus support many boundary conditions, e.g. copy, radiative, fixed, etc.
- Periodic boundary conditions have been supported since version 1 (contrary to rumour). There are thorns to support Cartoon boundaries and Rotational symmetries (in development), and we are working on making the symmetries completely transparent to simulation codes.

Current Capabilities: Interaction

- The HTTPD thorn provides an interface which allows a web-browser to connect to a running simulation
- This allows a user to examine the state of the running simulation and change certain parameters, such as the frequency of Io or the variables to be output, or in fact any parameter which some thorn author has declared may be changed during the simulation.
- These capabilities may be extended by any other thorn. E.g. the HTTPDExtra thorn allows the user to download any file output by the IO thorns in the Computational toolkit, and even to view two-dimensional slices as jpegs.
    - There is also a helper-script for web-browsers which allows the appropriate visualisation tool to be launched when a user requests a file.

Current Capabilities: Visualization

- The output from the Computational Toolkit IO thorns can be visualised by many clients, such as: Amira, OpenDX, GnuPlot, Xgraph, Ygraph, ...
- There is currently work from the climate modelling community to add IO thorns for the NetCDF format and this will bring in a new set of possible visualisation clients.
- The IsoSurfacer thorn calculates isosurfaces of a variable in parallel and may stream the data out to a suitable client.
- We provide one, **IsoView**, but it is an open format which can be used by other clients, e.g. **Amira**.
- Lots of information on web pages, including binaries for various visualisation tools and HOWTOs for setting these tools up and using them.

Cactus Summary

- Cactus is a powerful framework for developing portable applications, particularly suited to large collaborations. I've barely scratched the surface in this talk.
  - See http://www.cactuscode.org for more information.
- Cactus is currently used by many groups around the world, in several fields, and the number of users is growing.
- Cactus 4.0 was a major revamp of the infrastructure to make it cleaner and more modular.
- Future versions of Cactus will add features which increase the range of methods and problem domains which it is suited for.

Agenda

- Introduction to Cactus
  - What is Cactus
  - Flesh and thorns
  - Cactus Computational Toolkit
- Cactus Demo
  - WaveToy Demo

Questions

For more information:

www.cactuscode.org

Gabrielle Allen: gallen@cct.lsu.edu

Yaakoub El Khamra: yye00@cct.lsu.edu